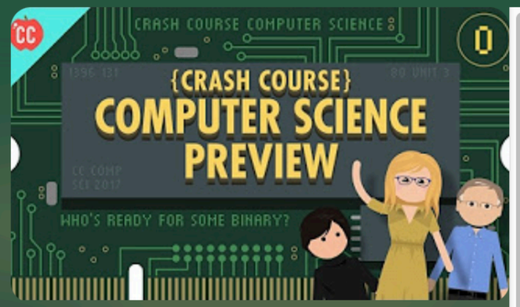
Computer Architecture

A gentle introduction

Josh Borrow, Jun 10th



Computer Science

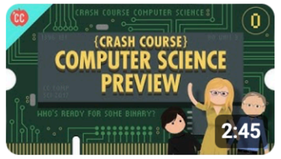
CrashCourse

41 videos 14,177,773 views Last updated on Mar 19,...

Play all Shuffle

In 40 episodes, Carrie Anne Philbin teaches you computer science! This course is based on introductory college-level material as well as the AP Computer Science Principles guidelines. By the end of this course, you will be able to:

*Outline the history of computers and the design decisions that gave us modern computers
*Describe the basic elements of programming and software
*Identify the basic components of computer hardware and what they do
*Describe how computers are used and how that has evolved over time

1 

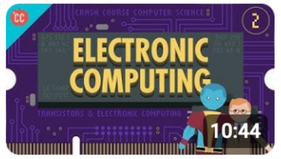
Crash Course Computer Science Preview

CrashCourse • 3.6M views • 7 years ago

2

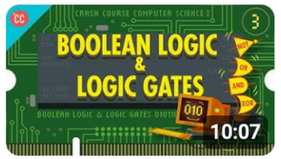
Early Computing: Crash Course Computer Science #1

CrashCourse • 3.8M views • 7 years ago

3 

Electronic Computing: Crash Course Computer Science #2

CrashCourse • 2.1M views • 7 years ago

4 

Boolean Logic & Logic Gates: Crash Course Computer Science #3

CrashCourse • 2.8M views • 7 years ago

5

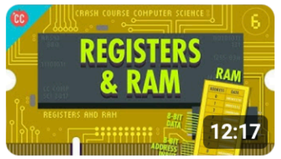
Representing Numbers and Letters with Binary: Crash Course Computer Science #4

CrashCourse • 1.8M views • 7 years ago

6

How Computers Calculate - the ALU: Crash Course Computer Science #5

CrashCourse • 1.9M views • 7 years ago

7 

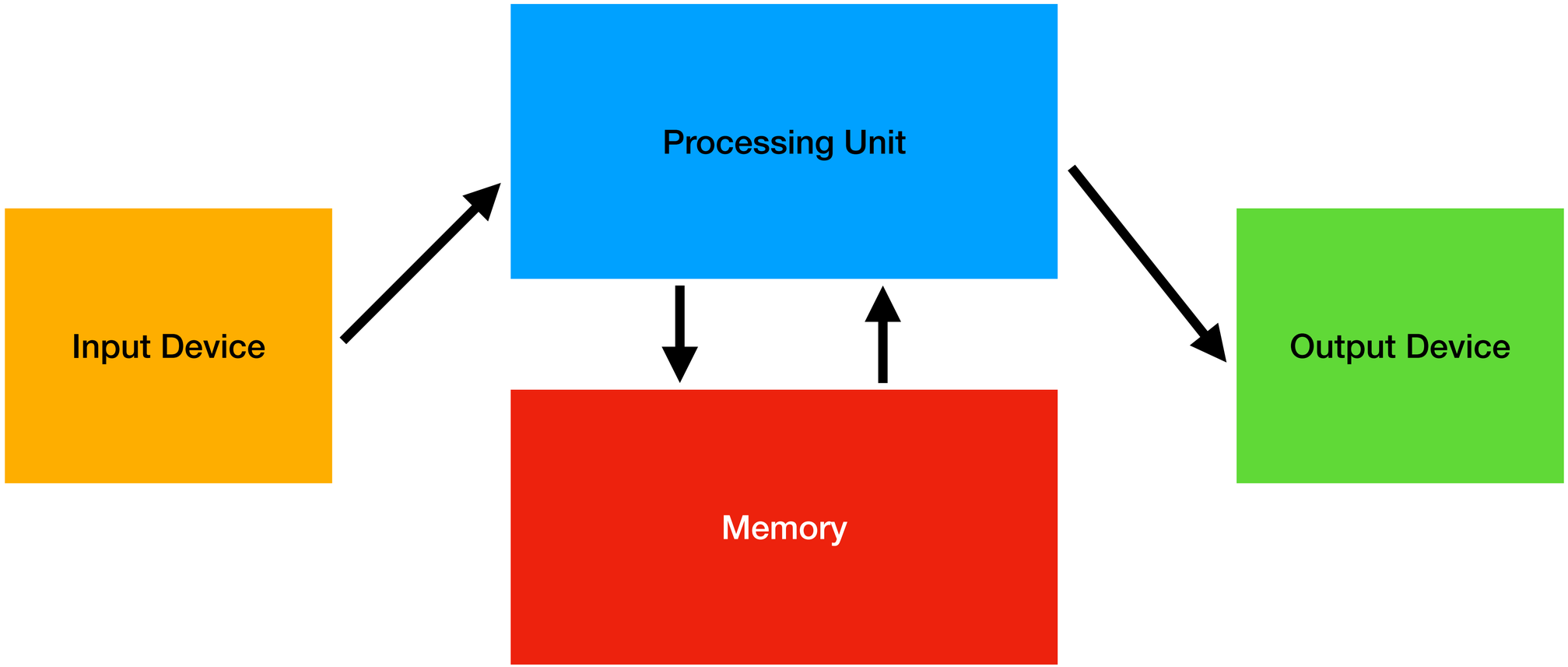
Registers and RAM: Crash Course Computer Science #6

CrashCourse • 2M views • 7 years ago

But… Software?

- As all your software runs on hardware, it’s crucial to understand it.
- Even a **base-level knowledge of computer architecture** can help you write vastly more performant programs.
- Knowing your hardware also comes in clutch when debugging programs.

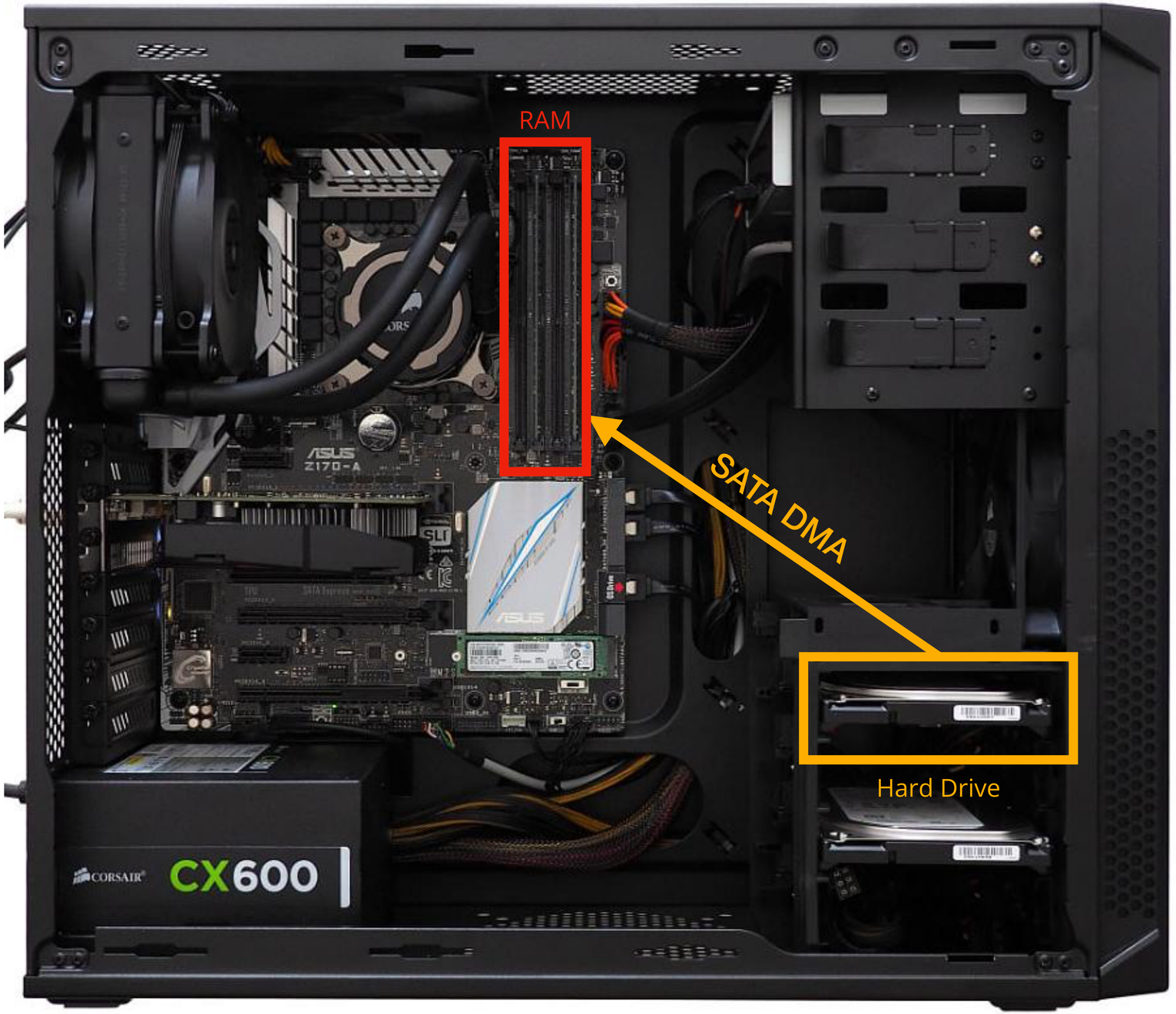
A basic ‘computer’

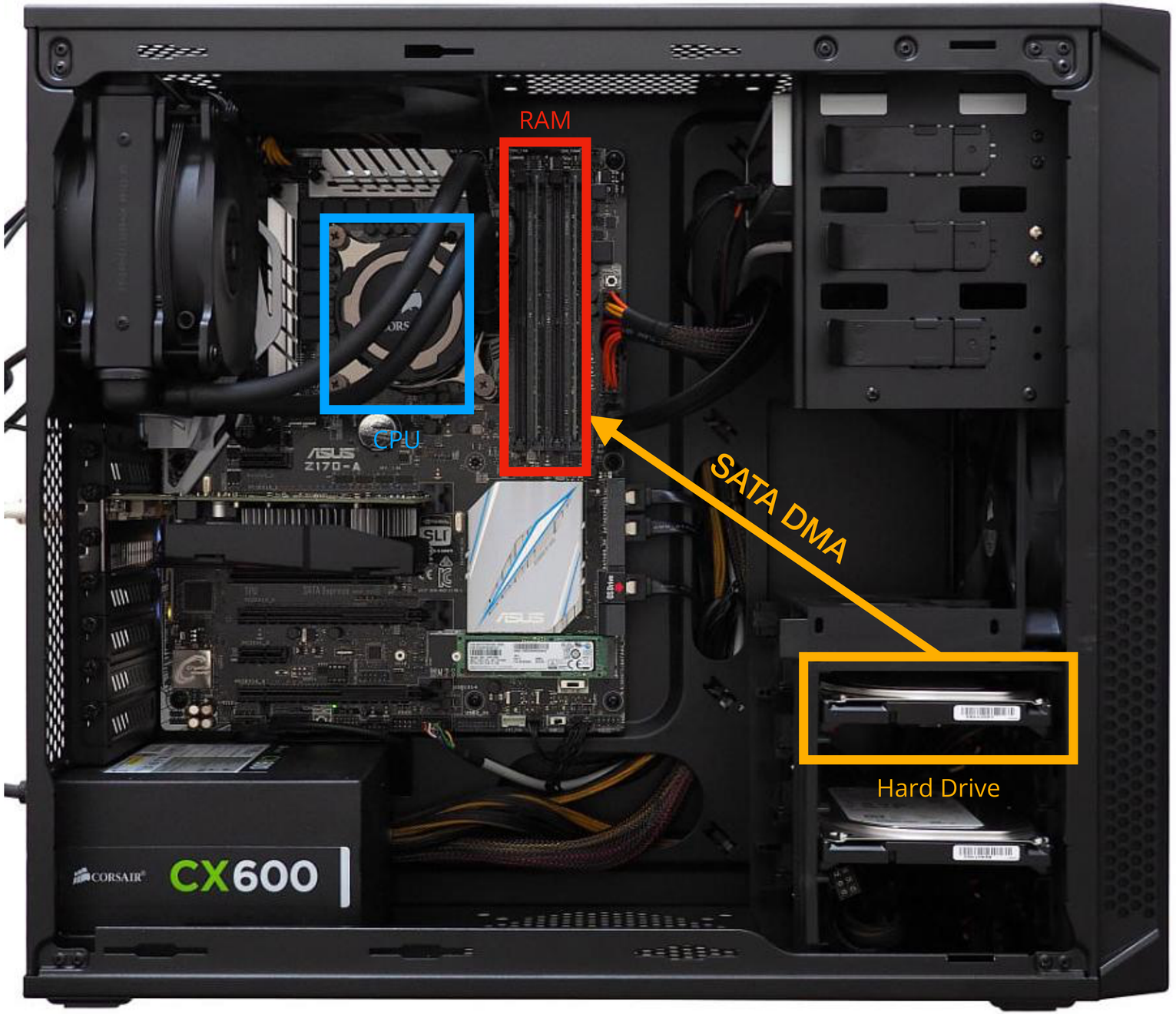


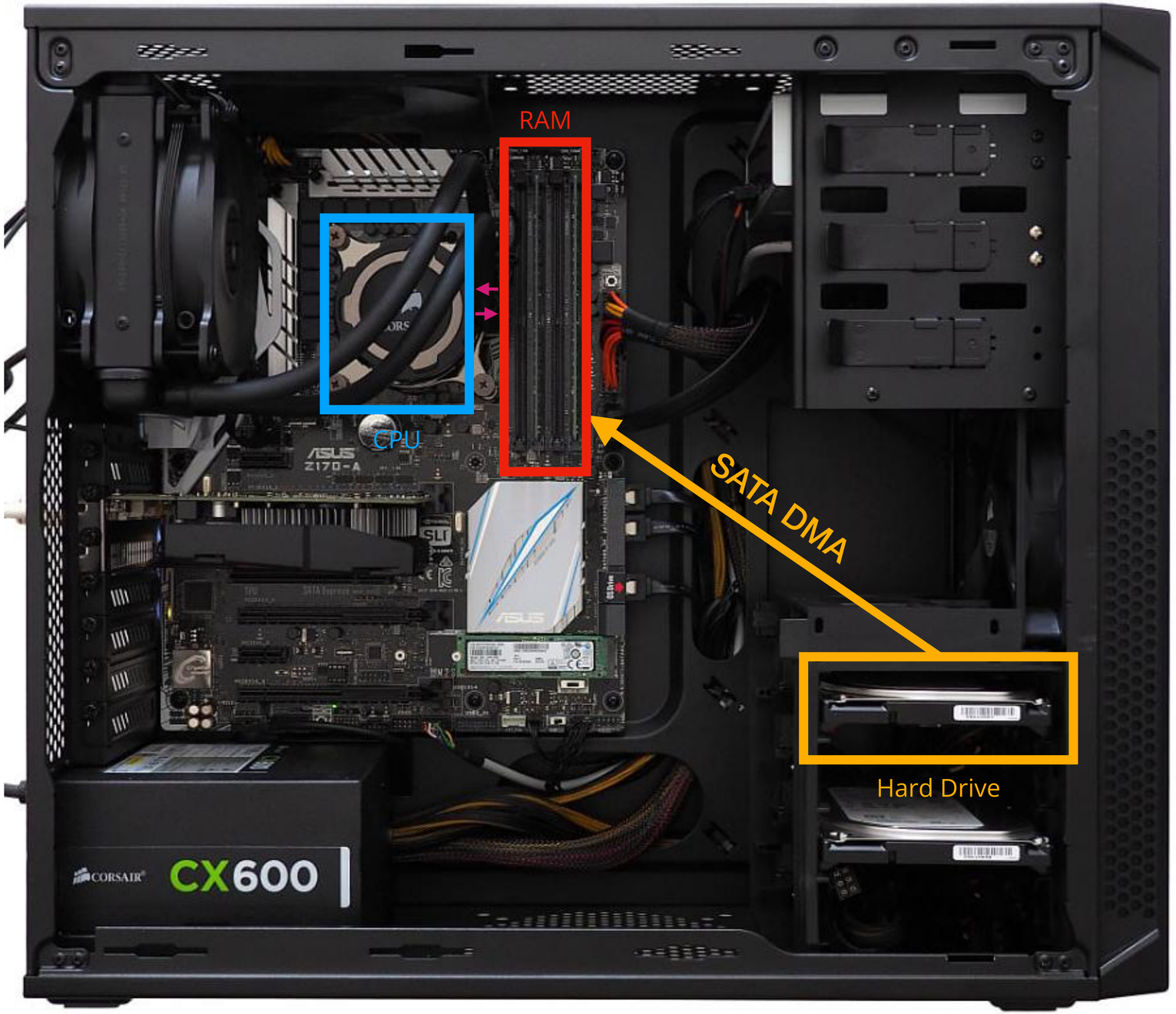
ASUS
Z170-A
CORSAIR
CX600

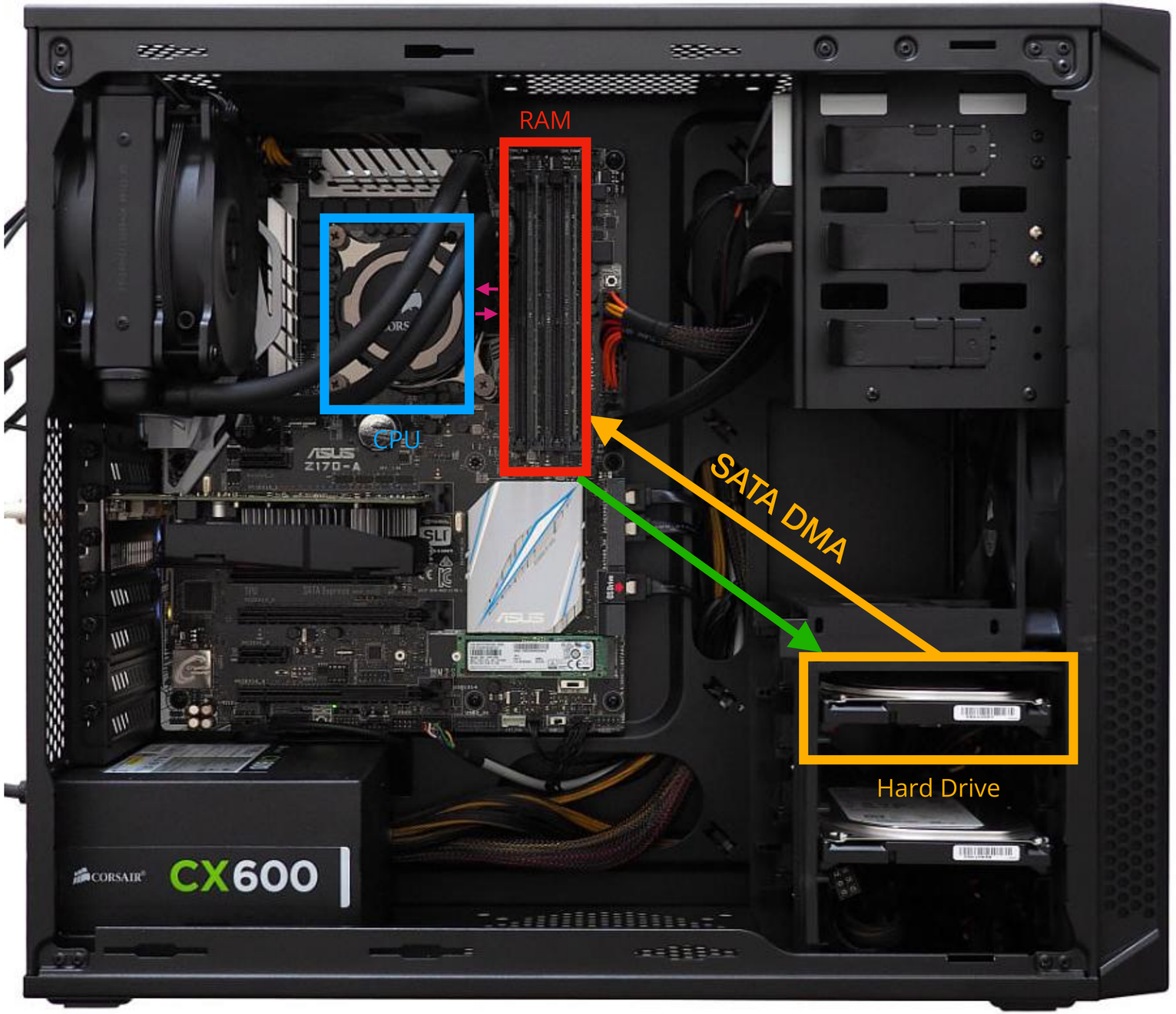
ASUS
Z170-A
CORSAIR
CX600
Hard Drive

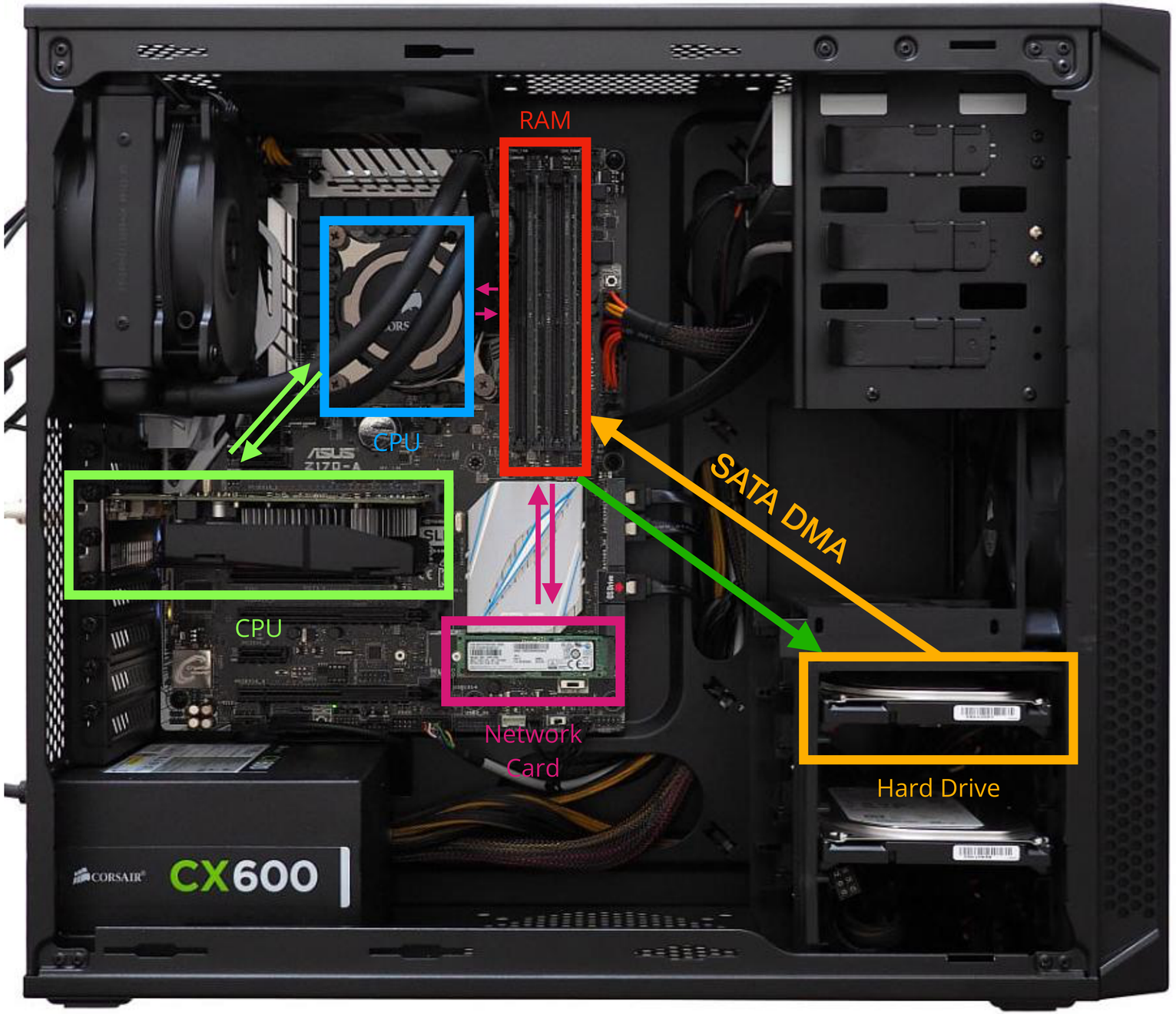
SATA DMA
Hard Drive
ASUS
Z170-A
CORSAIR
CX600


RAM
SATA DMA
Hard Drive
ASUS
Z170-A
CORSAIR
CX600


RAM
CPU
SATA DMA
Hard Drive
ASUS
Z170-A
CORSAIR
CX600


RAM
CPU
SATA DMA
Hard Drive
ASUS
Z170-A
CORSAIR
CX600

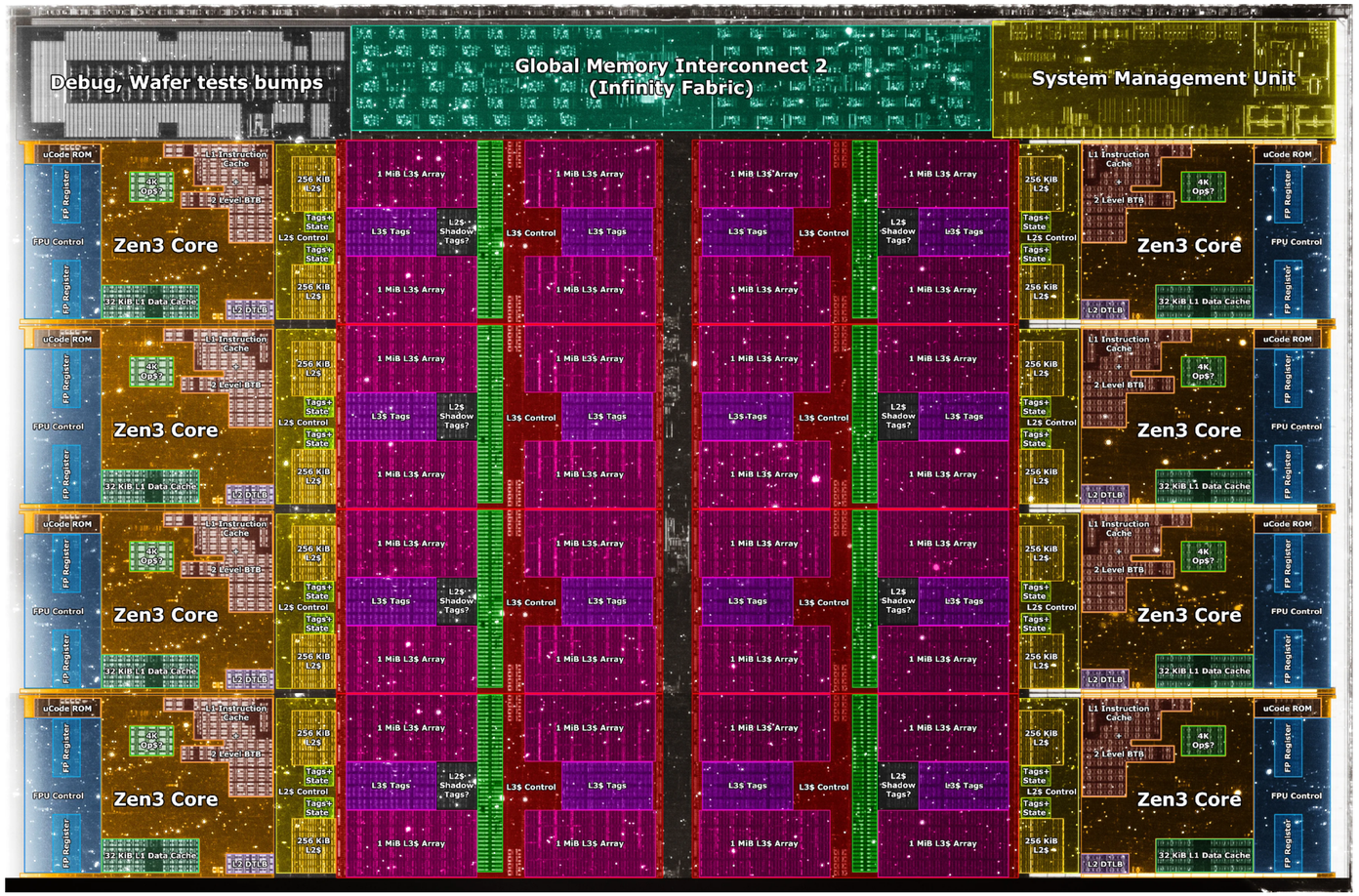

RAM
CPU
SATA DMA
Hard Drive
CORSAIR
CX600
ASUS
Z170-A


RAM
CPU
CPU
SATA DMA
Network
Card
Hard Drive
CORSAIR
CX600

Network
Node
CPU + Memory
Storage

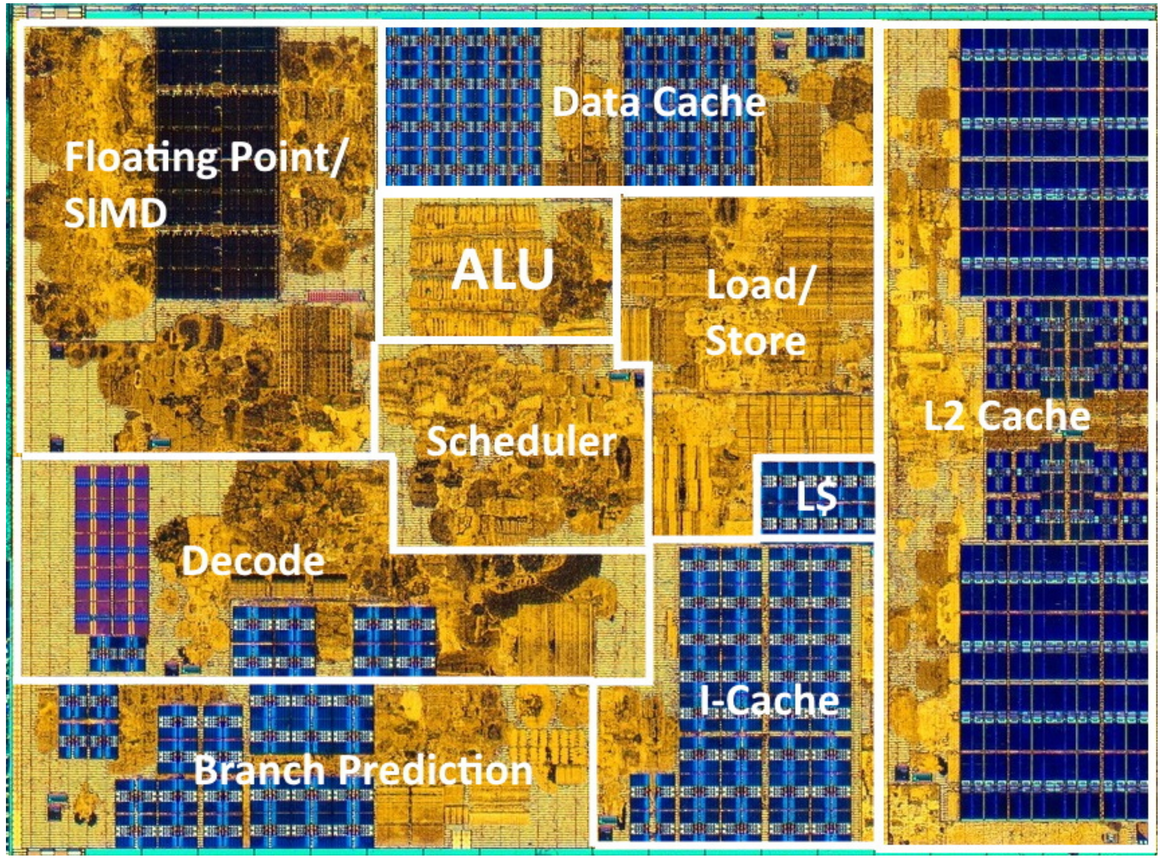
But what is the CPU?

- The CPU, at a hardware level, is a very messy **integrated circuit**!
- It has lots of components, but at the end of the day its main goal is to do math, as quickly as possible.
- **Modern CPUs** are made up of **'cores'**, which are **independent processing units** that can communicate (but do different things at the same time).



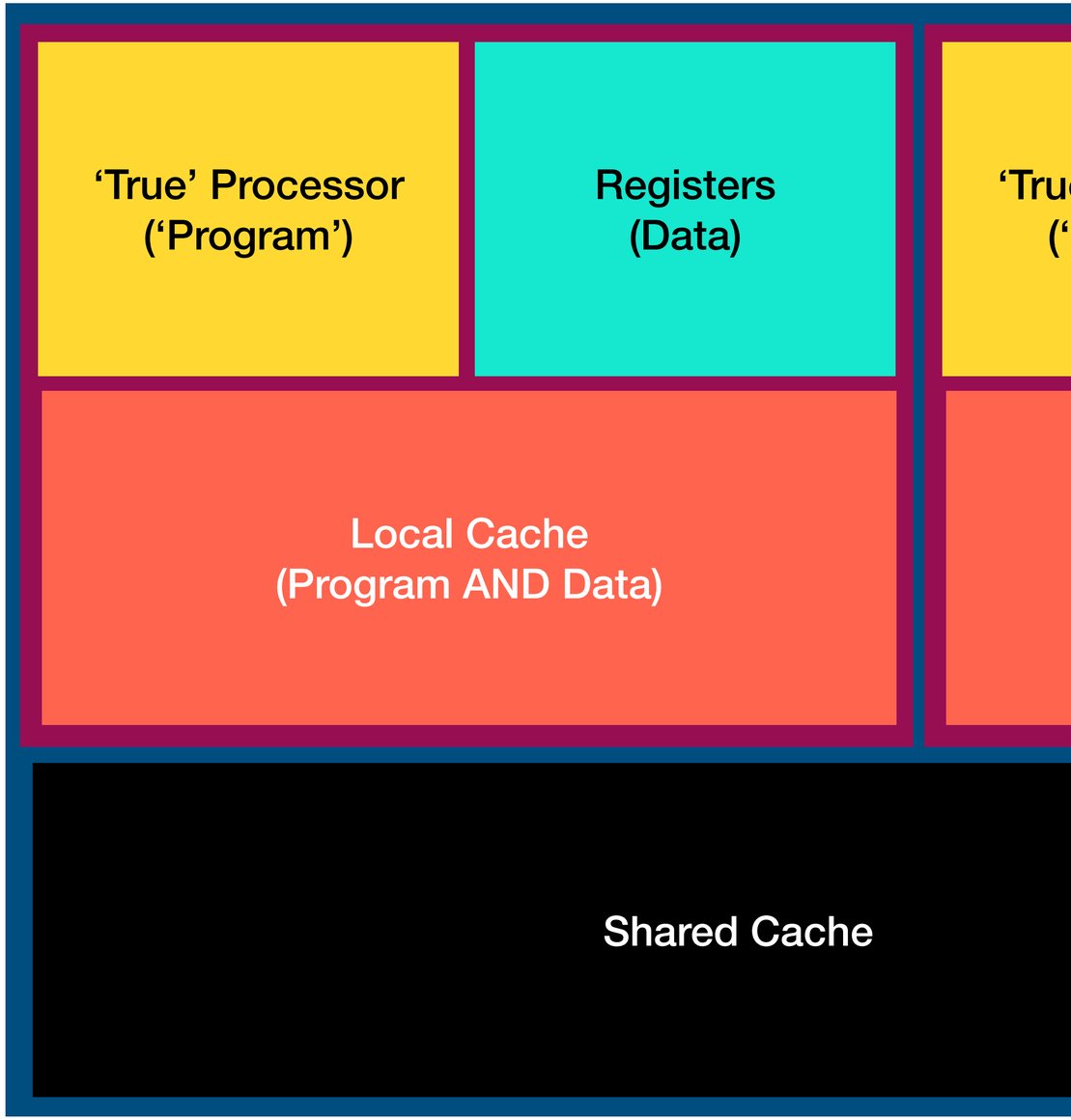
But what is the CPU?

- The CPU, at a hardware level, is a very messy **integrated circuit**!
- It has lots of components, but at the end of the day its main goal is to do math, as quickly as possible.
- **Modern CPUs** are made up of **'cores'**, which are **independent processing units** that can communicate (but do different things at the same time).



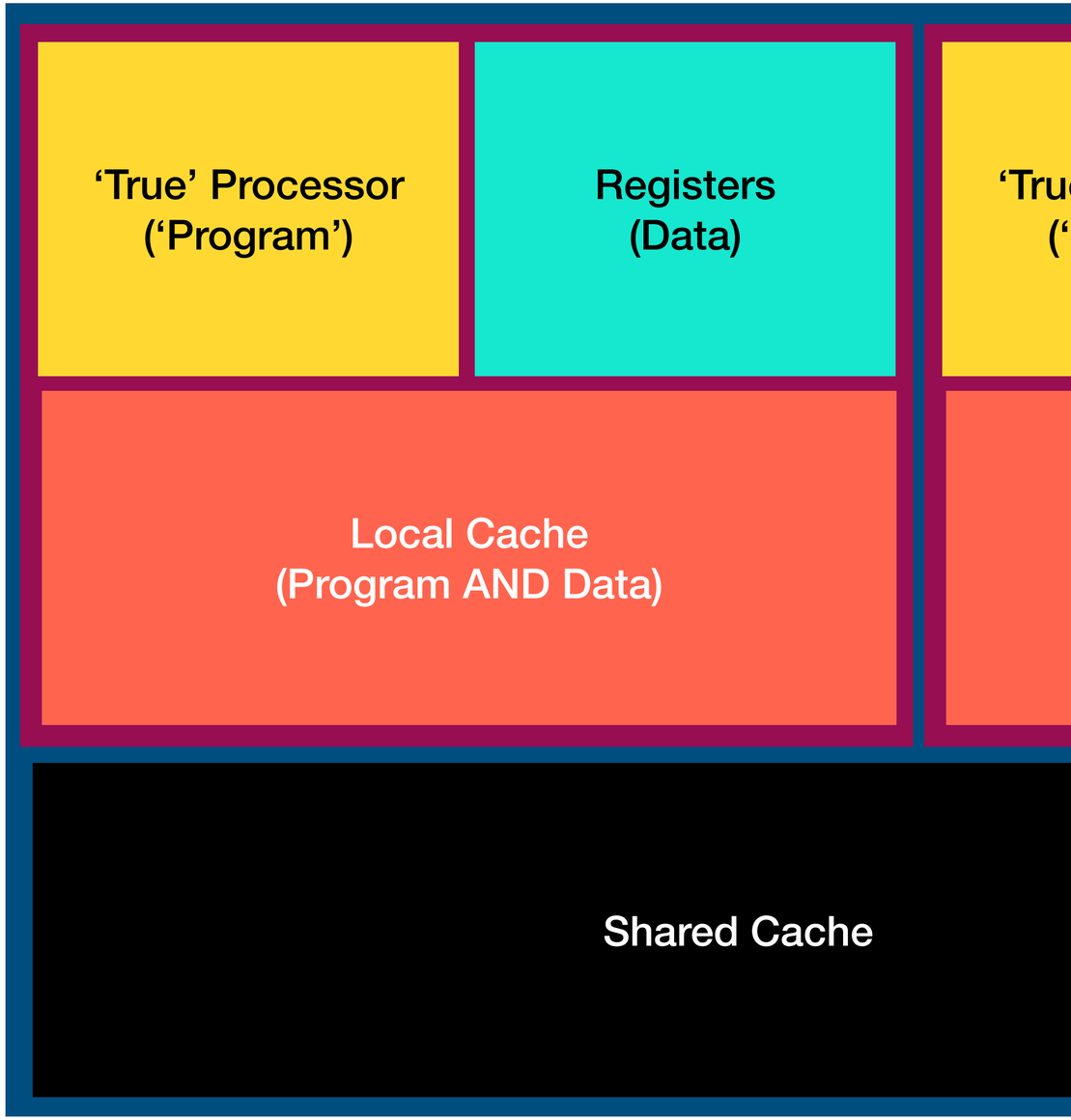
CPU layout

- The program you are trying to execute is, in its most base form, a **set of instructions** for the CPU to execute.
- Instructions are things like…
  - Add two numbers together
  - Multiply two numbers together
  - Move a piece of data from one location to another



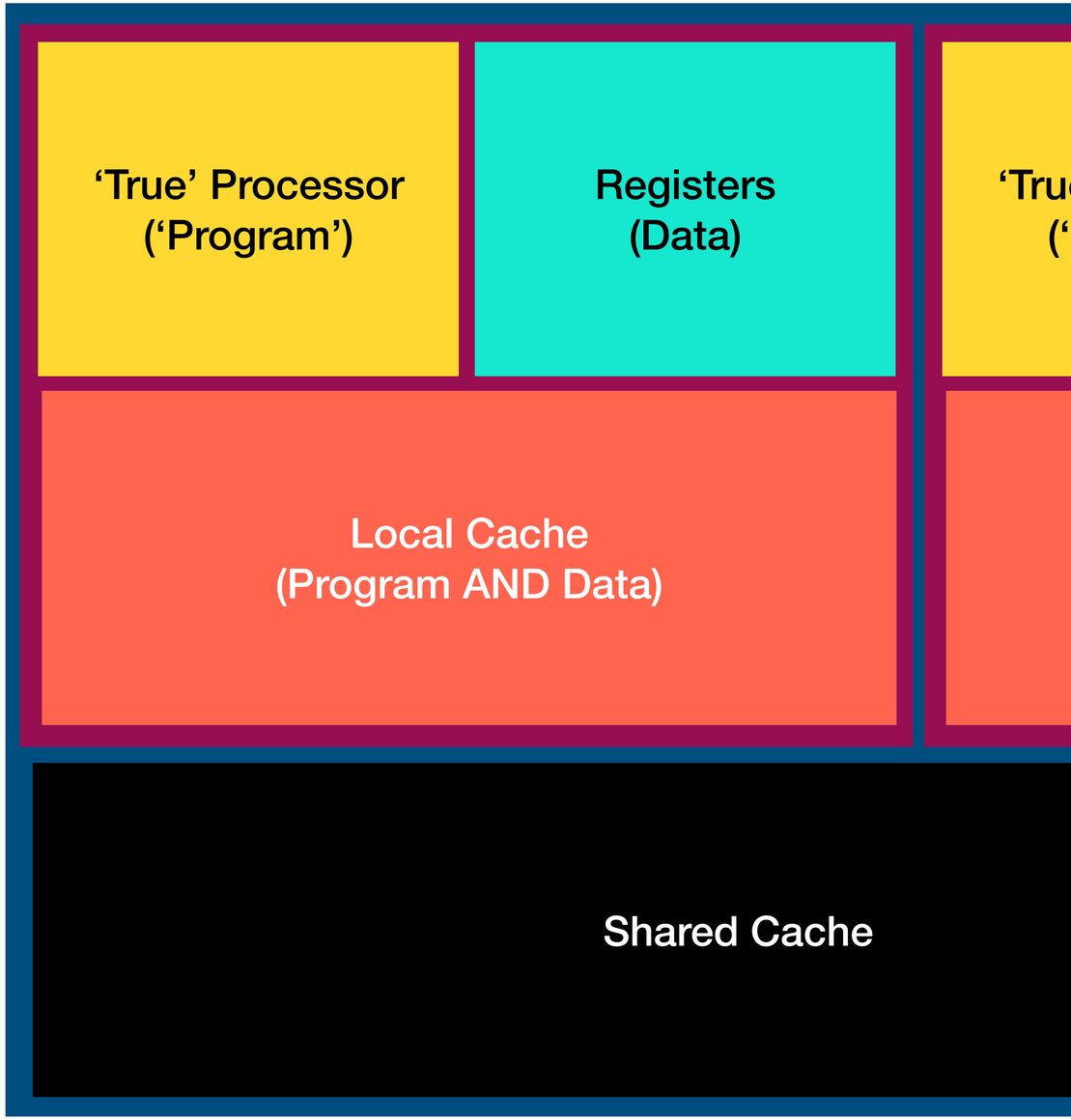
CPU layout

- In the CPU, there are various levels of '**cache**'. These are **small bits** of memory that are **extremely fast** and are co-located with the circuitry for doing the processing.
- Your **program and required data are loaded into the lowest level cache**. **This is** relatively **slow**. The CPU then asks for the next instruction, and it's go time!



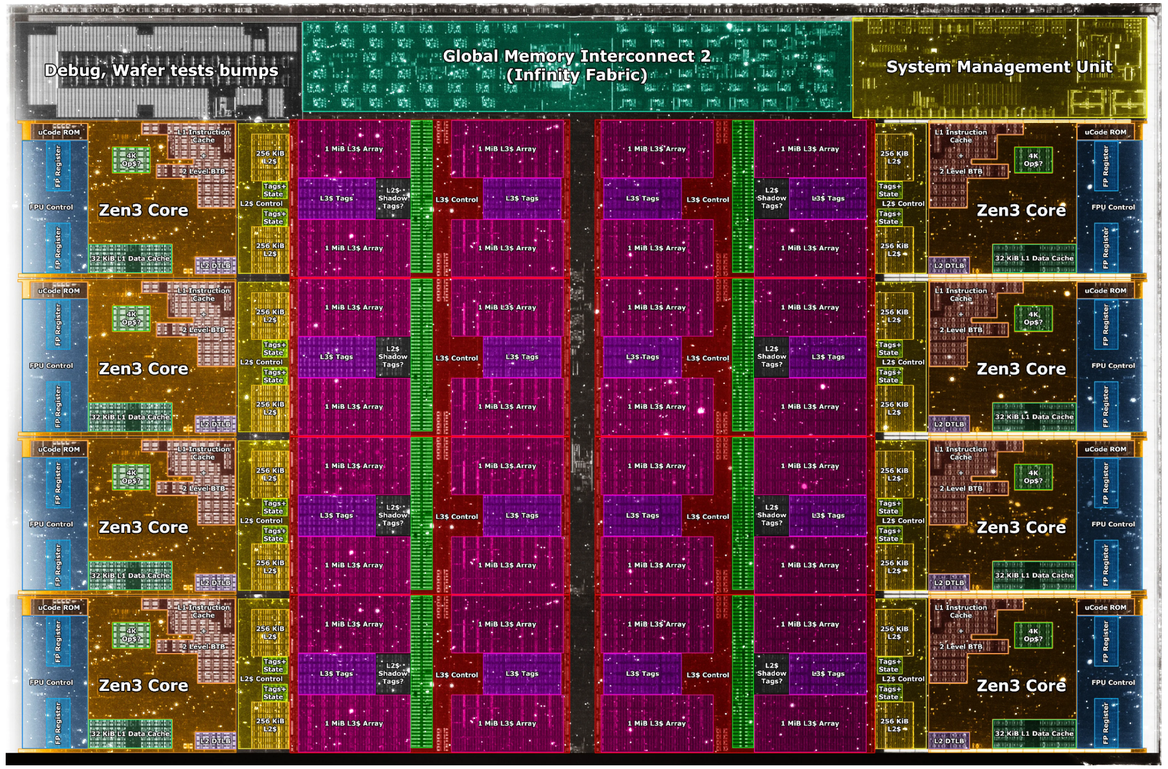
CPU layout

- Next to the CPU there are **registers**. These registers are **extremely fast** and well-connected data storage (there is space for about **700** numbers per core).
- Once you have completed your set of instructions, your **data needs to make its way out of registers back into main memory**, which is a **slow** process.



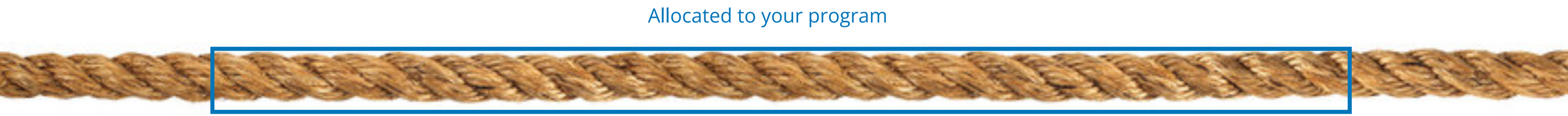
Multiple Cores

- Modern CPUs have **many** of these **processing units** all on one chip. Your laptop probably has around 4-12, and server CPUs have 32-128.
- All these cores **share resources**, though (e.g. they have a shared cache, interfaces with main memory), which can **cause contention**!



But what is RAM?

But what is RAM?



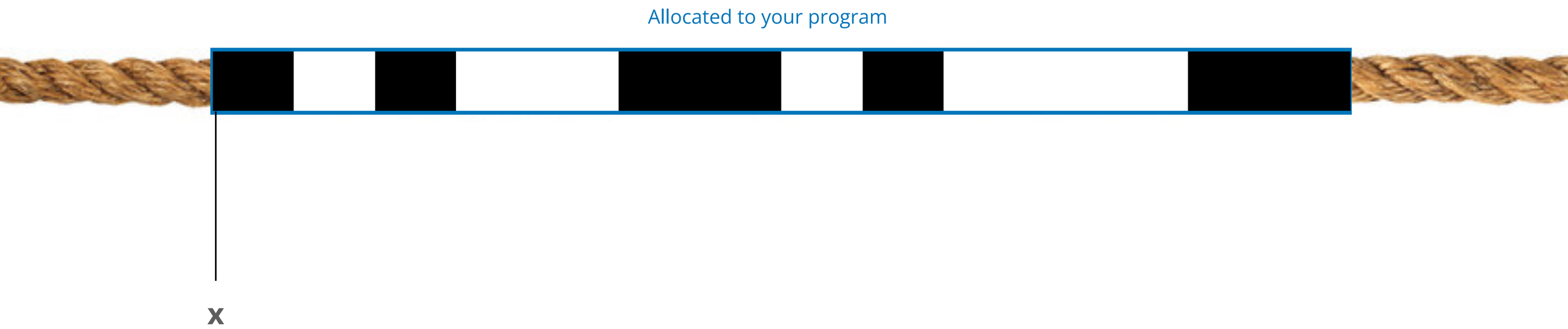
But what is RAM?

Allocated to your program

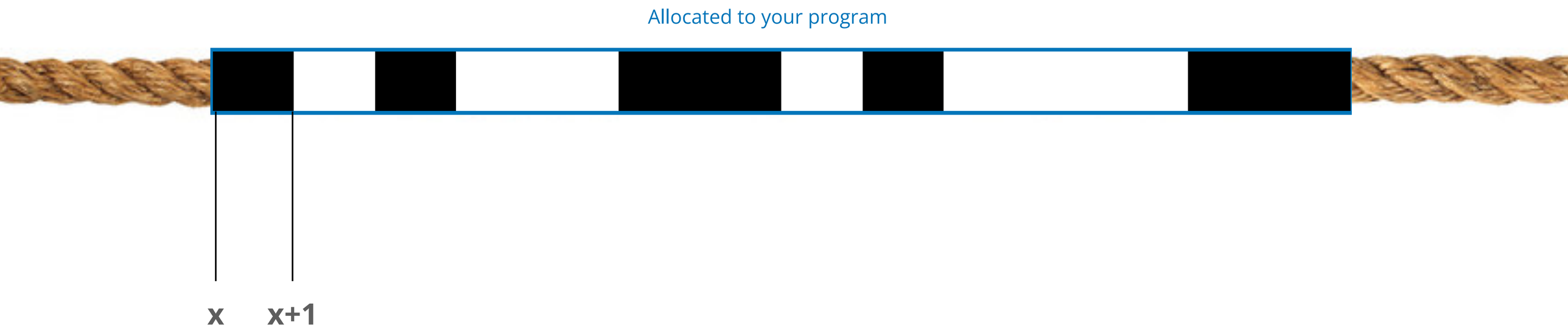
But what is RAM?

Allocated to your program

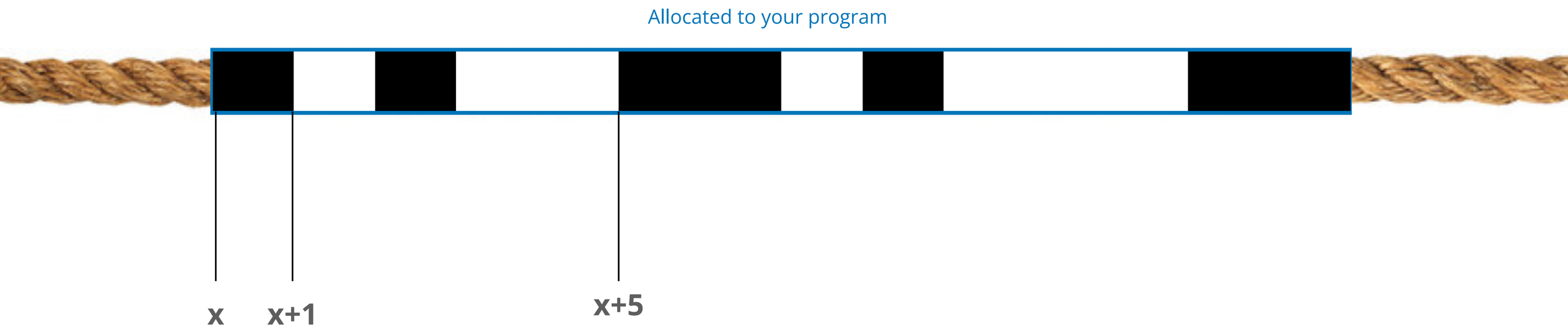
But what is RAM?



But what is RAM?



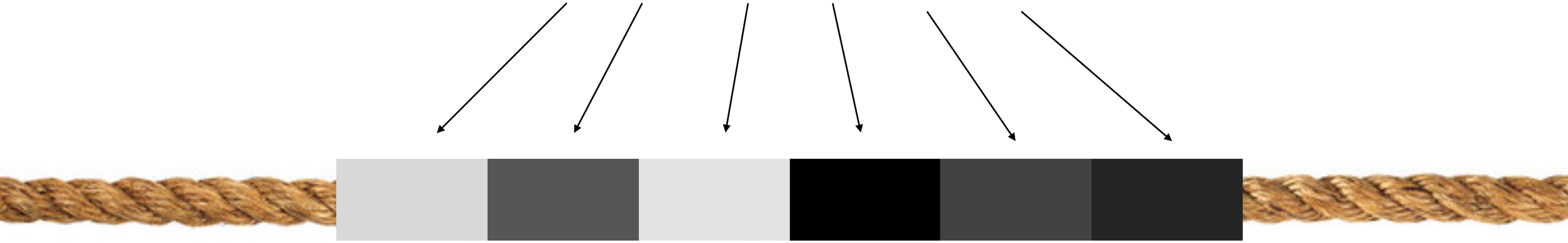
But what is RAM?



Arrays in Memory

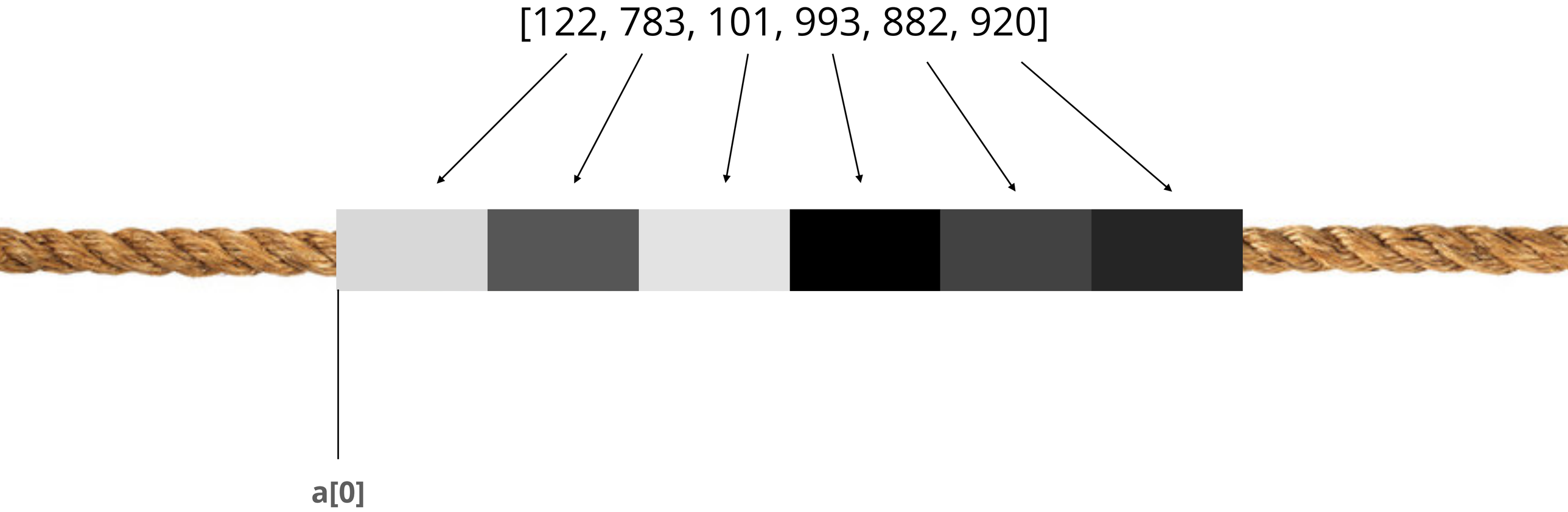
- Arrays (e.g. numpy arrays) are maximally space efficient structures!

[122, 783, 101, 993, 882, 920]



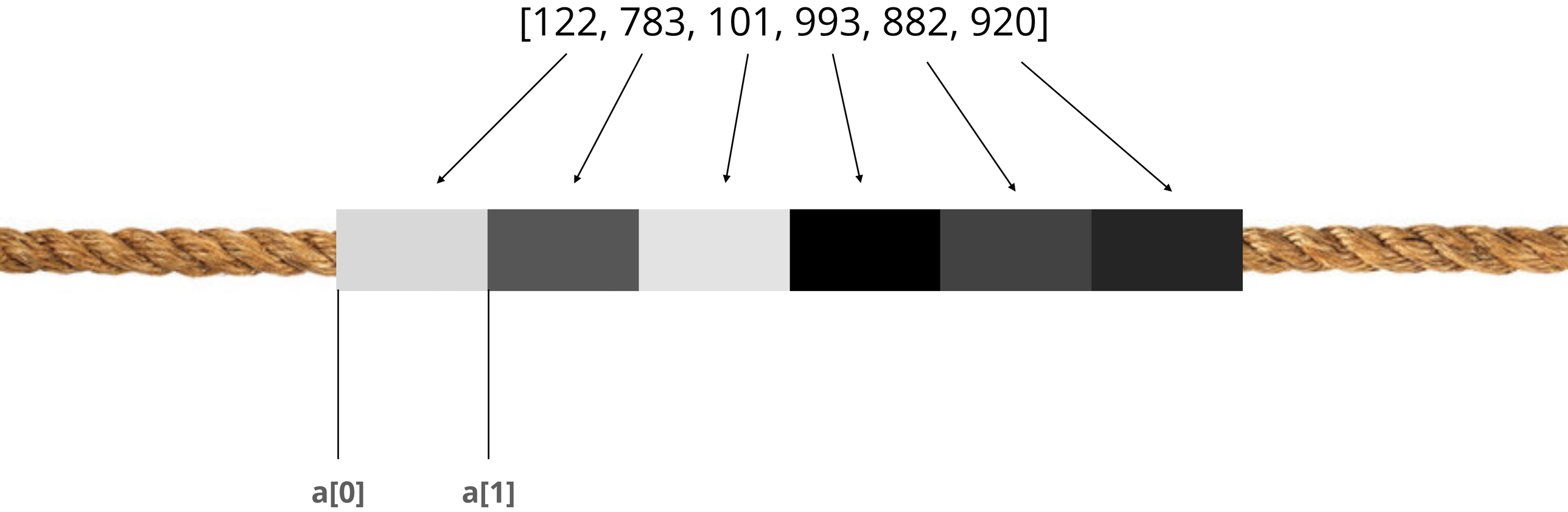
Arrays in Memory

- Arrays (e.g. numpy arrays) are maximally space efficient structures!



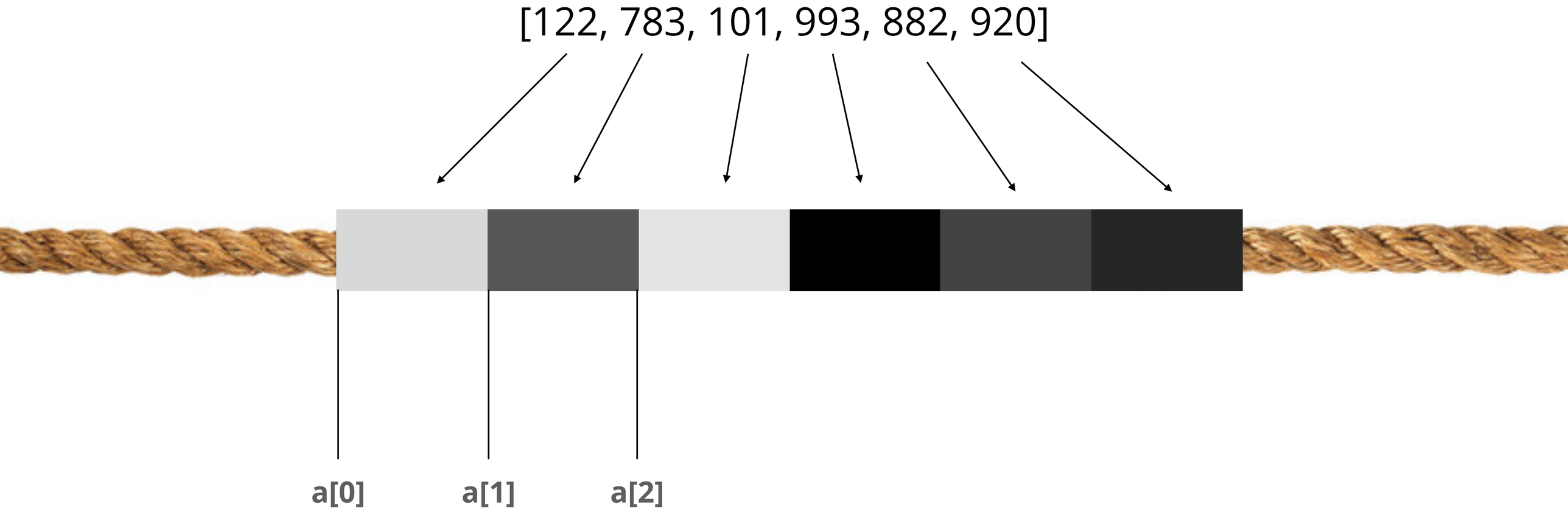
Arrays in Memory

- Arrays (e.g. numpy arrays) are maximally space efficient structures!



Arrays in Memory

- Arrays (e.g. numpy arrays) are maximally space efficient structures!



N-dimensional arrays?

- What about 2D arrays..?
- We just stack them together!

[122, 783, 101, 993, 882, 920]

[250, 750, 130, 299, 899, 999]

...

N-dimensional arrays?

- What about 2D arrays..?
- We just stack them together!

[122, 783, 101, 993, 882, 920]

...

N-dimensional arrays?

- What about 2D arrays..?
- We just stack them together!

N-dimensional arrays?

- What about 2D arrays..?
- We just stack them together!

[122, 783, 101, 993, 882, 920]

...

N-dimensional arrays?

- What about 2D arrays..?
- We just stack them together!

[122, 783, 101, 993, 882, 920]

[250, 750, 130, 299, 899, 999]

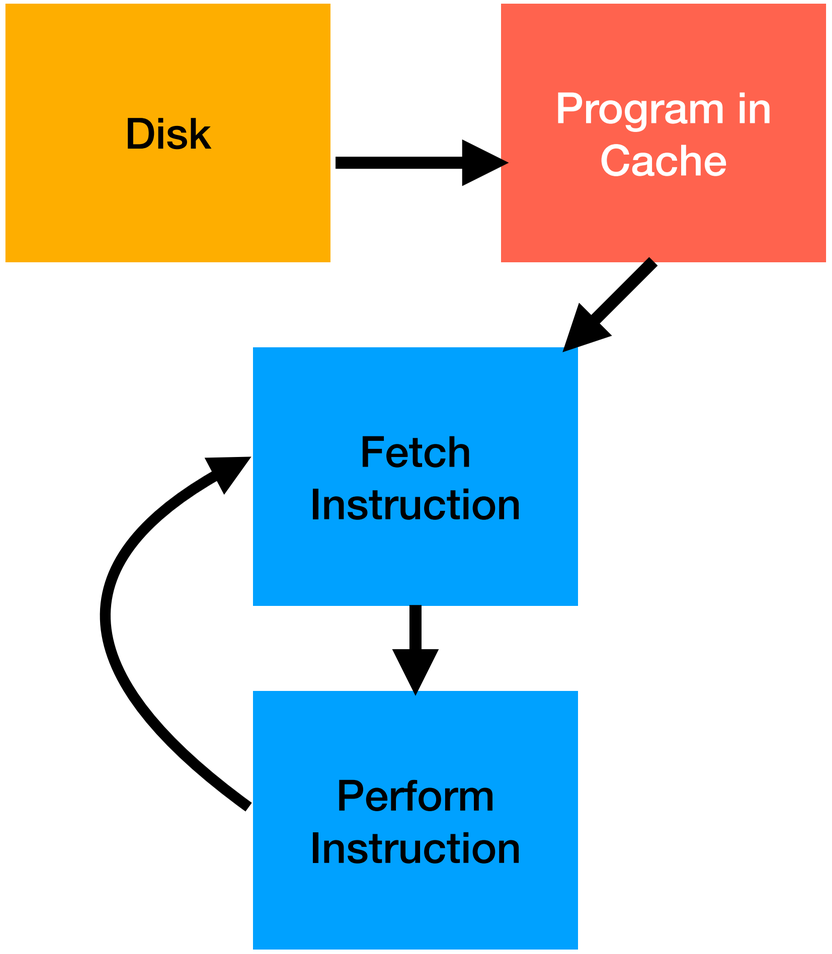
...

Your program

- Before we talk about how your program actually runs, we need to make a distinction between two different types of software:
  - Compiled software, written in languages like C
  - Interpreted software, written in languages like python

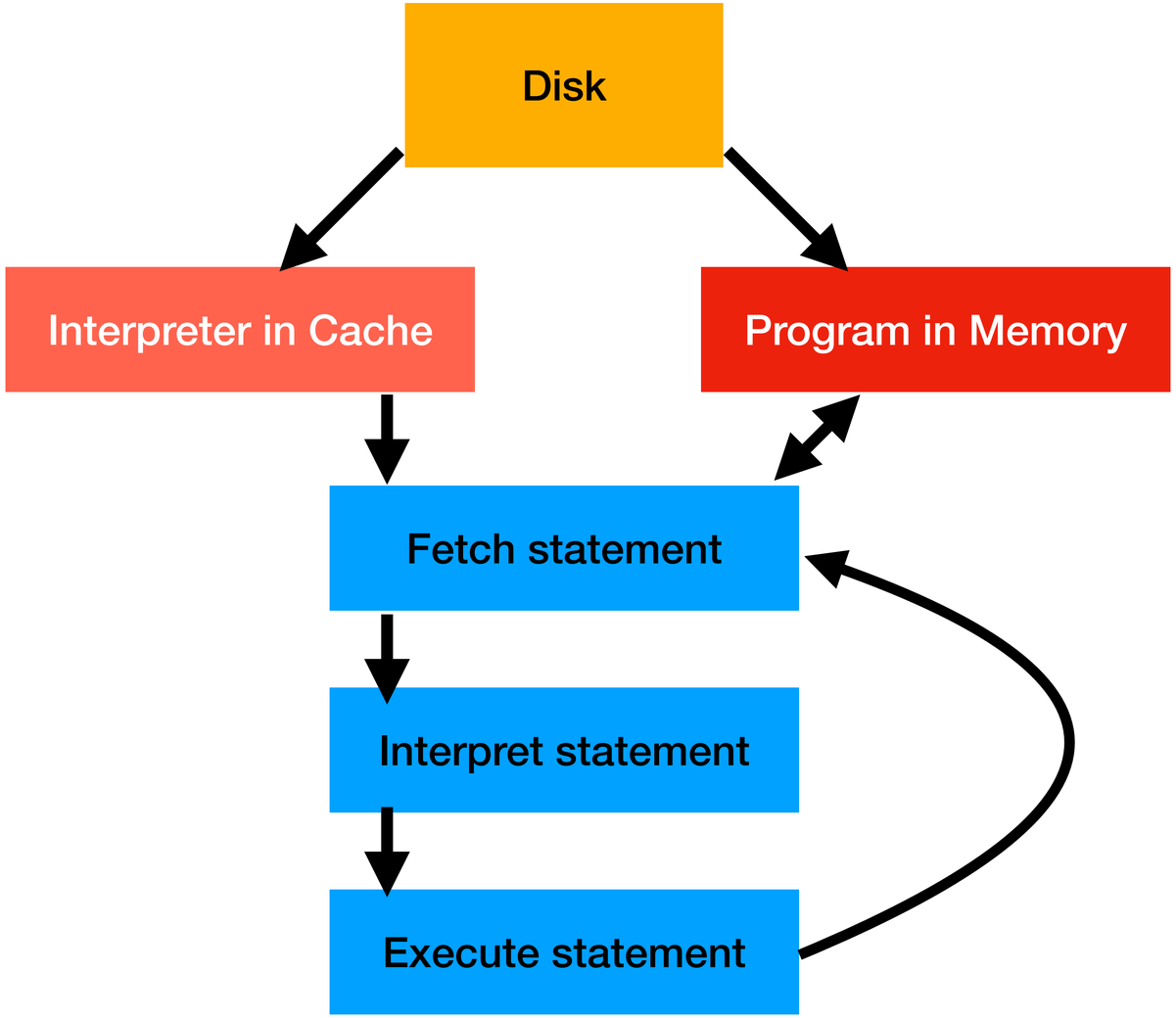
Compiled programs

- A ‘compiler’ turns your **entire** human-readable code into ‘**machine code**’ ahead of time.
- This machine code is **optimized** by the compiler ahead of time, and is hence usually uses **minimal operations** to perform tasks.
- **Execution flow** hence can then take a ‘**simple**’ path through the hardware.



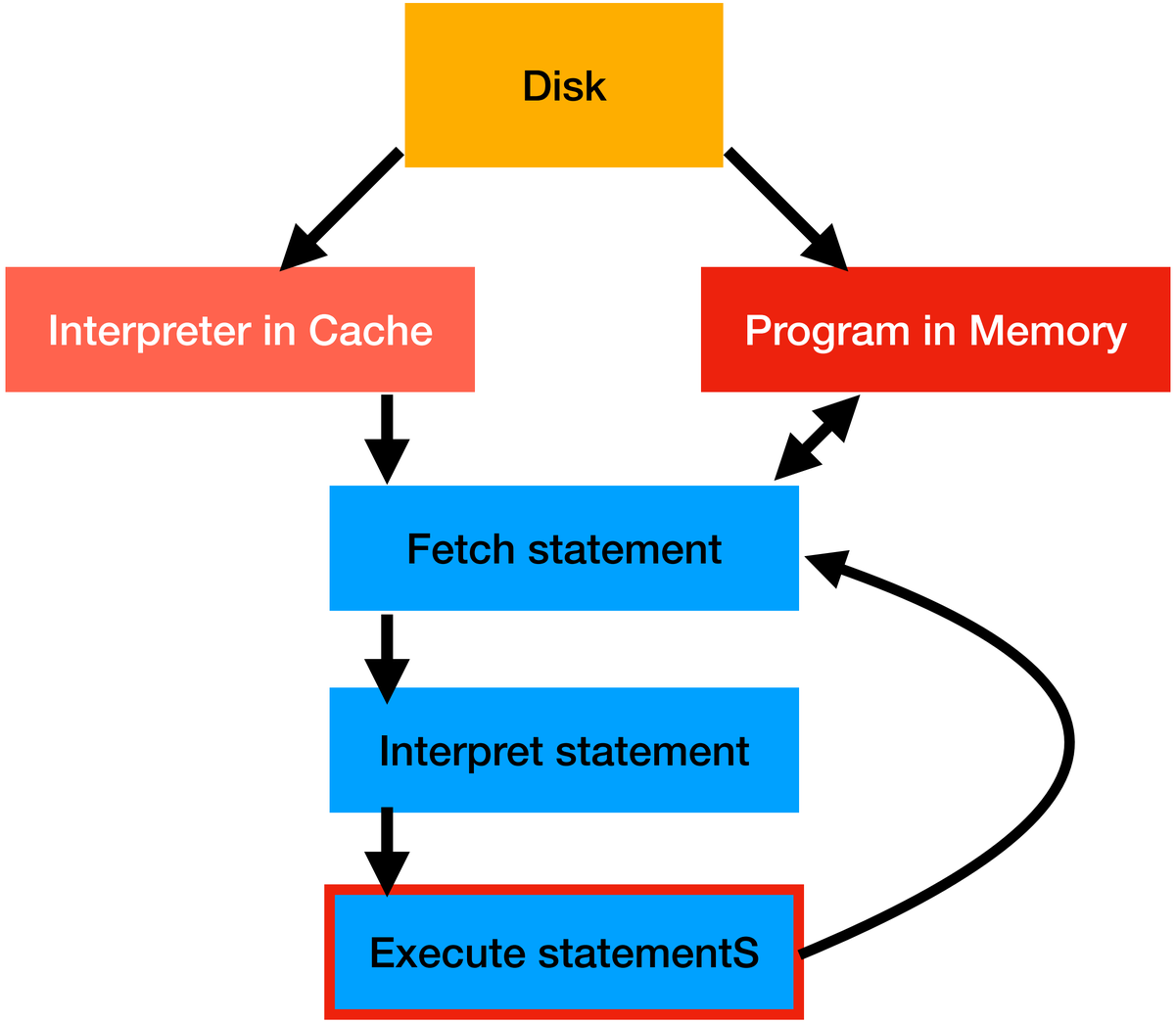
Interpreted programs

- In this case we **piggyback** on another program (the **interpreter**, e.g. CPython for python code) that manages our instructions.
- This leads to a **much more complex** (and hence slower) internal execution flow, due to **constant fetches from main memory**!



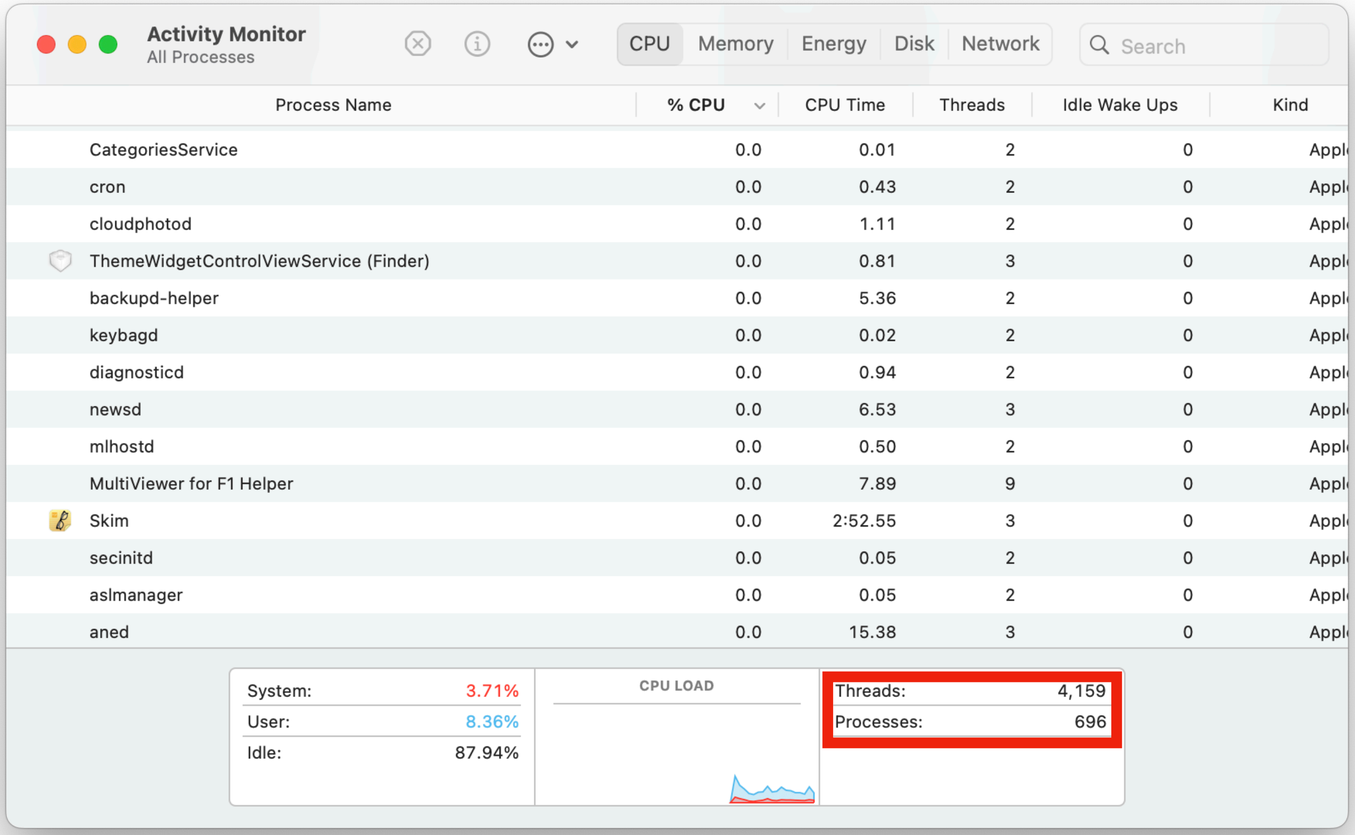
Compiled extensions

- With compiled extensions like numpy, you can '**vectorize**' the underlying operations.
- This **speeds up** code by mixing the flexibility of the interpreted code with the heavy use of caches by the compiled code, by **reducing reads from main memory**!



Being managed by the operating system

- Your programs always run in the context of the operating system.
- Have you ever wondered how you can have **more apps open than cores**?
- All modern operating systems use '**preemptive multitasking**'…

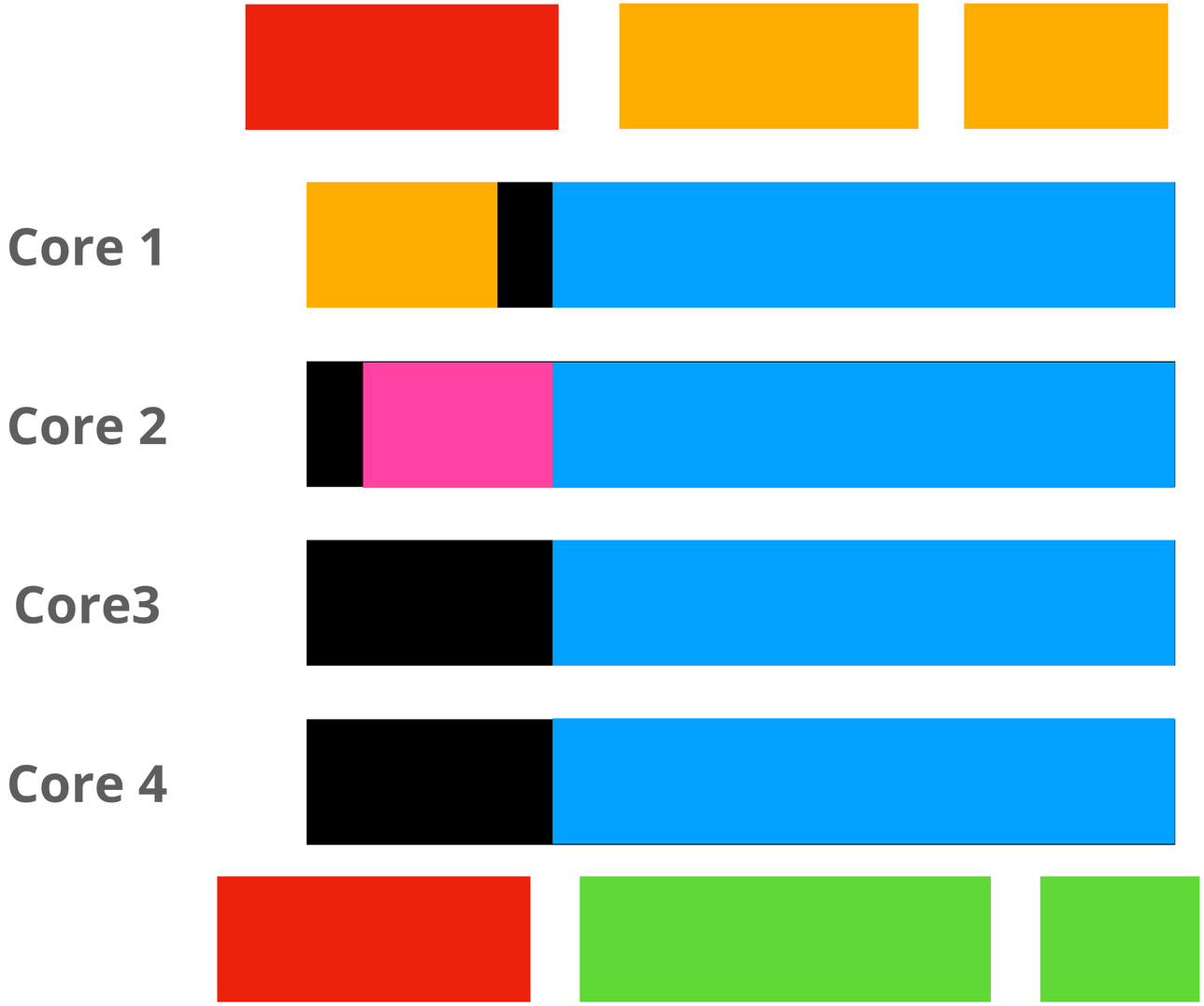


Being managed by the operating system

- Your programs always run in the context of the operating system.
- Have you ever wondered how you can have **more apps open than cores**?
- All modern operating systems use '**preemptive multitasking**'...

Being managed by the operating system

- The **OS can't steal control back as easily when you are deep in loops**; the CPU will simply ignore the OS.
- You must then be careful when running multi-threaded programs on your laptop; **don't assign them to all your cores**!



Summary

- A computer takes inputs, stores them in memory, performs processing on them, and writes them to an output.
- In a standard computer, the disk is the storage, RAM is the memory, and the CPU does the processing.
- Modern CPUs have multiple layers of caching and many cores that try to help them be as fast as possible.
- RAM can be thought of as a continuous string that stores information.
- Interpreted languages like python have compiled extensions to help extract good performance whilst maintaining flexibility.